ECSE-323
Digital System Design

Sequential Testing Lecture #2

Even with scan-path methods, the testing of modern chips and systems is very costly and time-consuming.

One approach to reducing the cost and complexity of external testers is to put circuitry on-chip to do a lot of the testing.

This approach is known as

Built-In Self-Test (BIST)

Testing difficulties addressed by BIST

- ***Increasing chip logic-to-pin ratio* – harder observability**
- ***Increasingly dense devices and faster clocks***
- ***Increasing test generation and application times***
- ***Increasing size of test vectors stored in ATE***
- ***Expensive ATE needed for 1 GHz clocking chips***
- ***Hard testability insertion* – designers are unfamiliar with gate-level logic, since they design at behavioral level**
- ***In-circuit testing* no longer technically feasible**
- ***Shortage of test engineers***
- ***Circuit testing cannot be easily partitioned***

BIST is used widely in modern chips

From the Pentium Developers Manual (ca 1997)

11.1. BUILT-IN SELF-TEST (BIST)

Self-test is initiated by driving the INIT pin high when RESET transitions from high to low.
No bus cycles are run by the Pentium processor during self-test.
The duration of self-test is approximately *2^19* core clocks. **[2.6 msec with 200MHz clock]**
Approximately *70%* of the devices in the Pentium processor are tested by BIST.
…
Upon completion of BIST, the cumulative result of all tests are stored in the EAX register. If EAX contains 0h, then all checks passed; any non-zero result indicates a faulty unit. Note that if an internal parity error is detected during BIST, the processor will assert the IERR# pin and attempt to shutdown.

What can be put on-chip?

Test vector generation
- FDTS test vector tables
- random test vector generation

- **Test vector presentation**
  - scan-path control
- **Test response analysis**
  - FDTS response tables
  - Response signature analysis

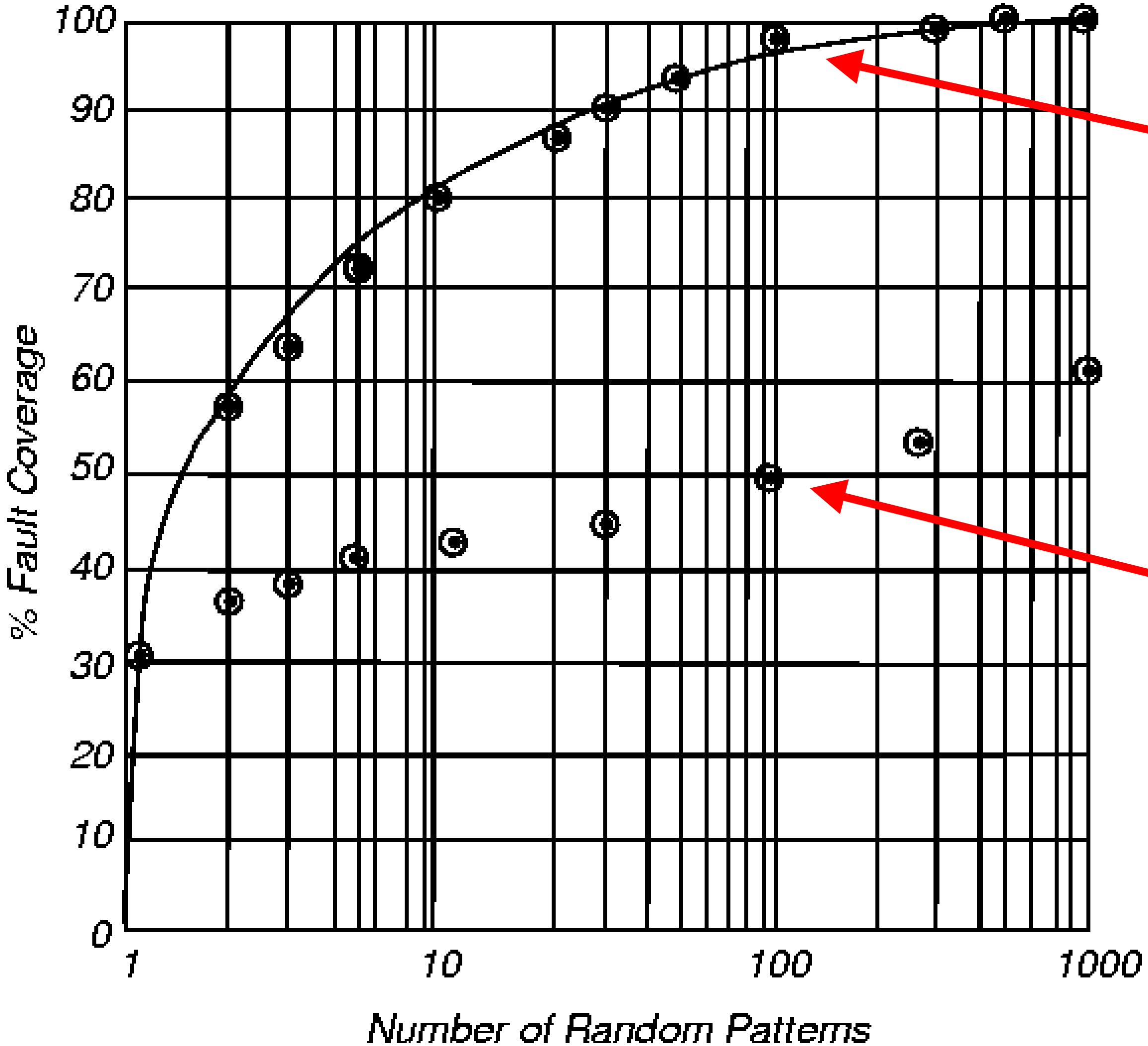
For circuits with large numbers of inputs and outputs, storage of the FDTS and test responses on-chip is impractical.

Thus, the generation of ***random*** test vectors is preferred.

The idea behind using random vectors is that a given vector of them will probably cover some of the faults.

If you have enough test vectors then you have a high probability of covering any fault.

The problem is that you need to generate a ***large number*** of random test vectors to achieve a good fault coverage level.



(a) Top curve -- random pattern testing with acceptable fault coverage.
(b) Bottom curve -- unacceptable random pattern testing.

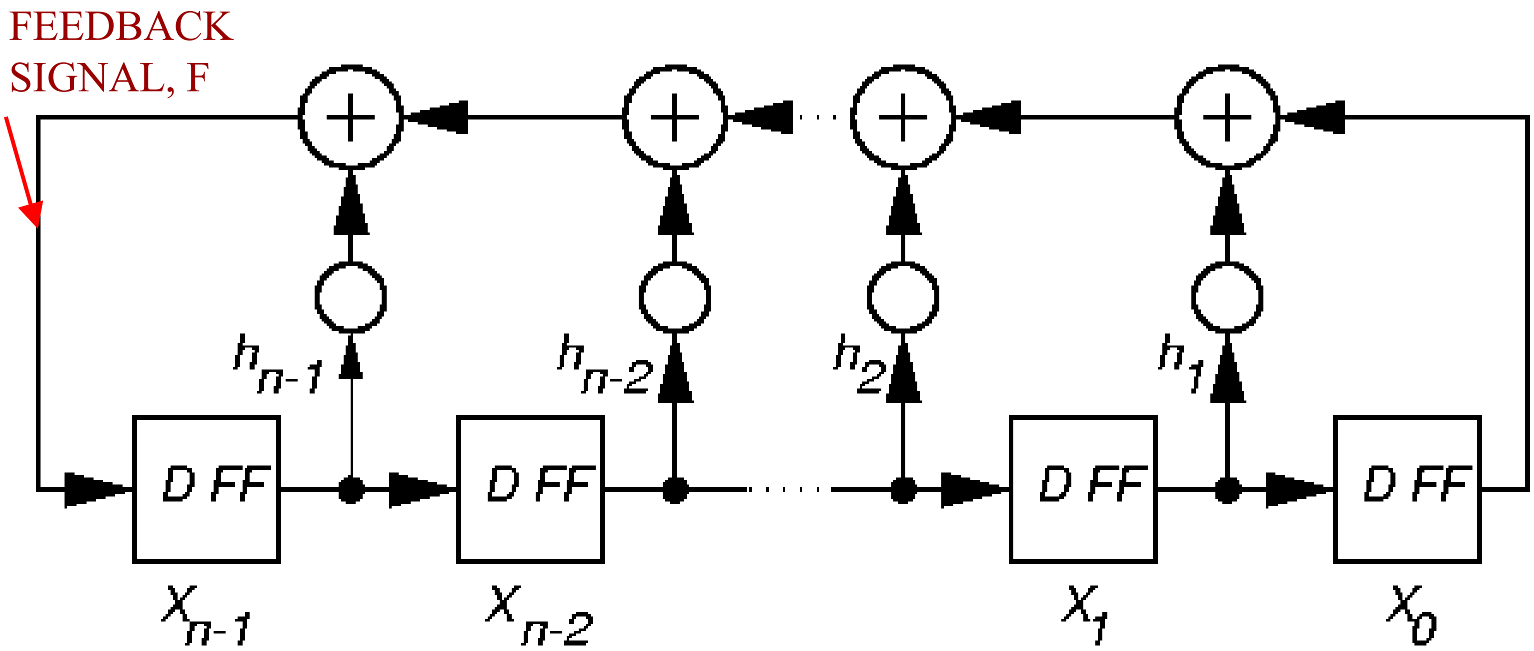
Some circuits are better suited to testing with random test vectors than others.

How can we generate random test vectors on-chip?

There is a very simple circuit that can generate ***pseudo-random*** bit streams.

This circuit is the ALFSR

(*__A__utonomous __L__inear __F__eedback __S__hift __R__egister*)



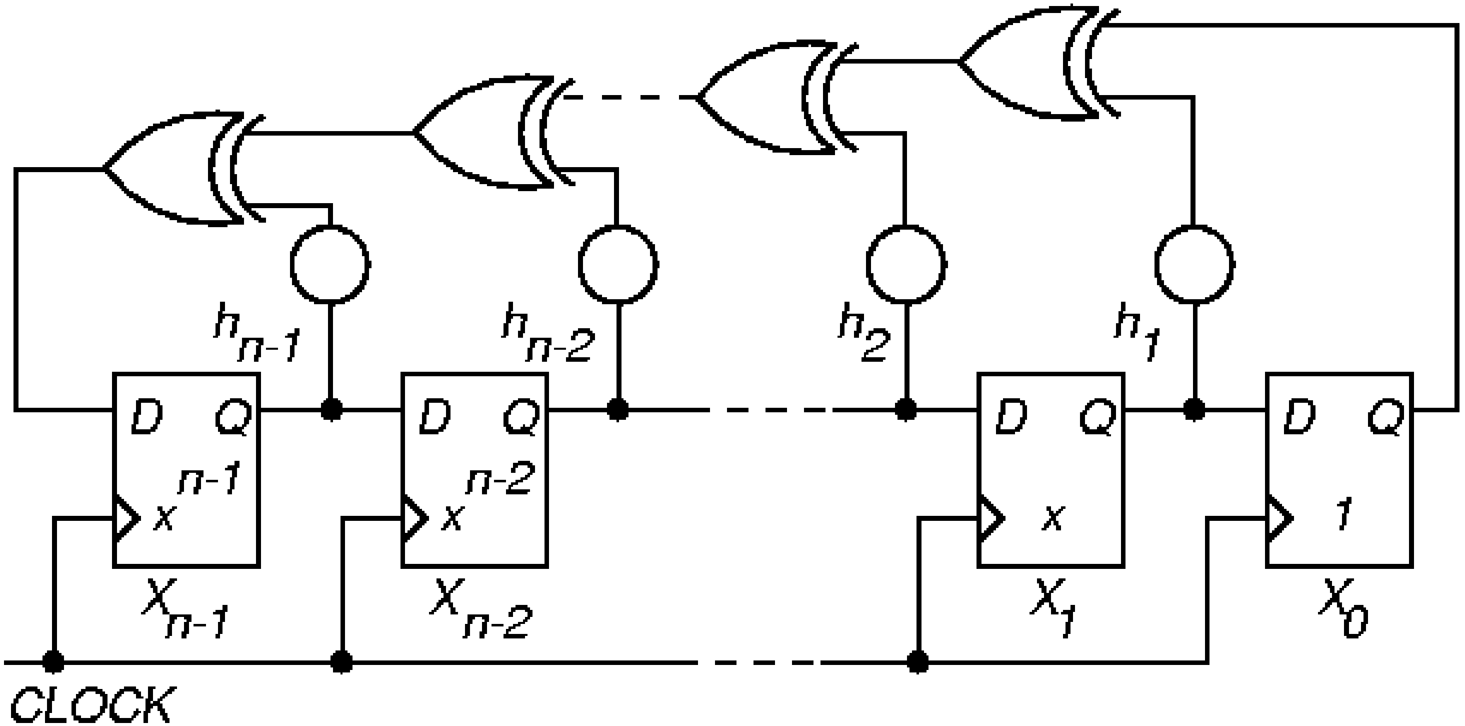
$$F = \sum_{i=0}^{n-1} X_i h_i = \sum_{i=0}^{n-1} F z^{i-n} h_i$$

or

$$F\Big(\underbrace{\sum_{j=0}^{n} z^{-j} h_{n-j}}_{p(z^{-1})}\Big) = 0, \quad h_0 = h_n = 1$$

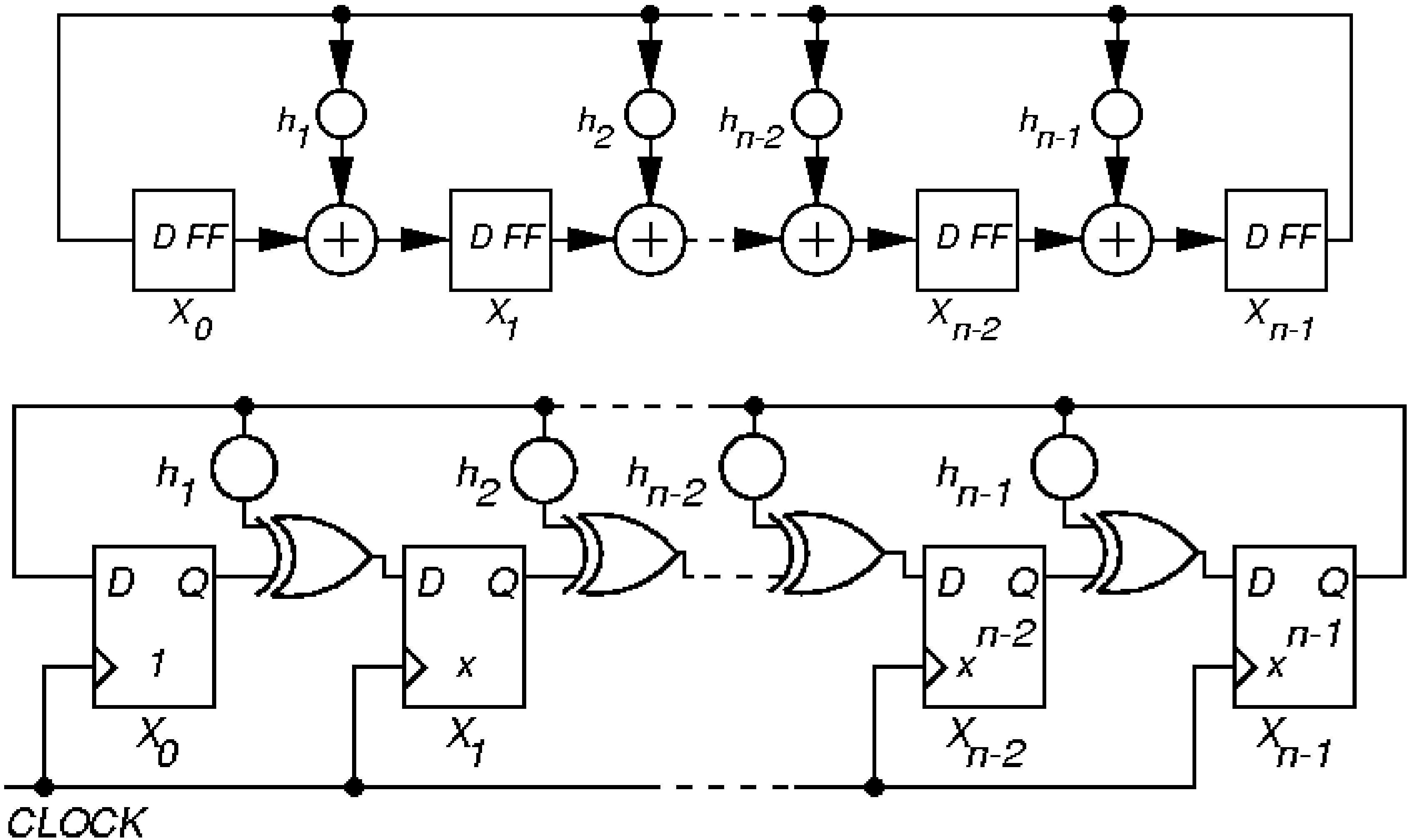
z^{-1} is the DELAY operator : the output of a flipflop, Q, is related to its input D by $Q = z^{-1}D$

$p(z^{-1})$ is known as the characteristic polynomial



In modulo-2 arithmetic, multiplication is done with the AND operation and addition with XOR

The AND is usually replaced with a direct connection or open circuit depending on the value of h



A faster implementation of an ALFSR

Certain cases of the characteristic polynomials, known as ***primitive polynomials***, result in ***maximal-length*** sequences.

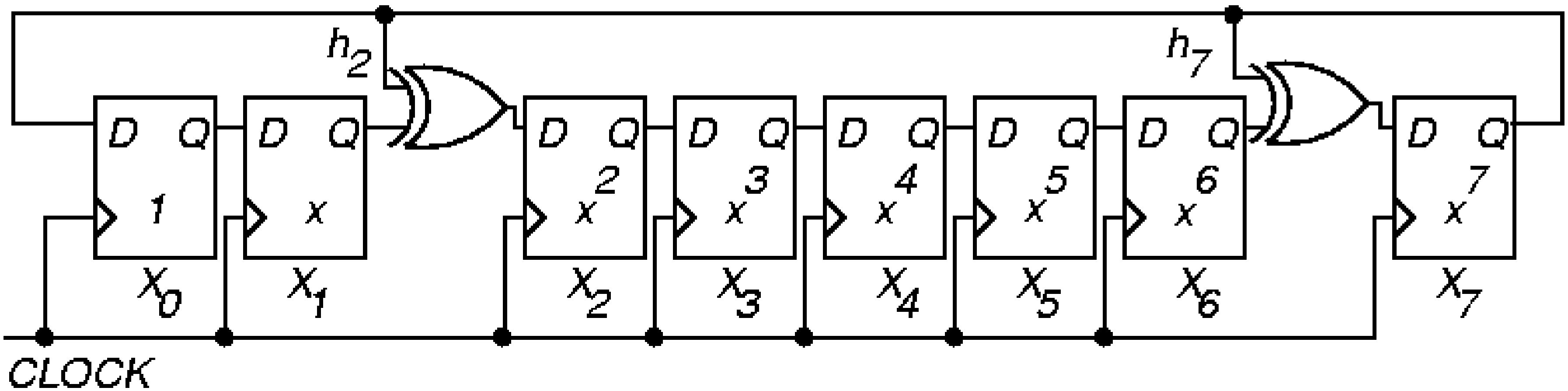
These sequences have length (or period) $2^N - 1$ (note: the sequence does not contain 0)

For example, an 8-bit maximal-length ALFSR would have a period of 255 clock cycles before the pseudo-random pattern repeats.
(*about 10 usec at 25 MHz*)

What would you guess as the period for an ALFSR that is **84 bits** long, if we clock the ALFSR at **25 MHz**?

Answer: 24 Billion Years

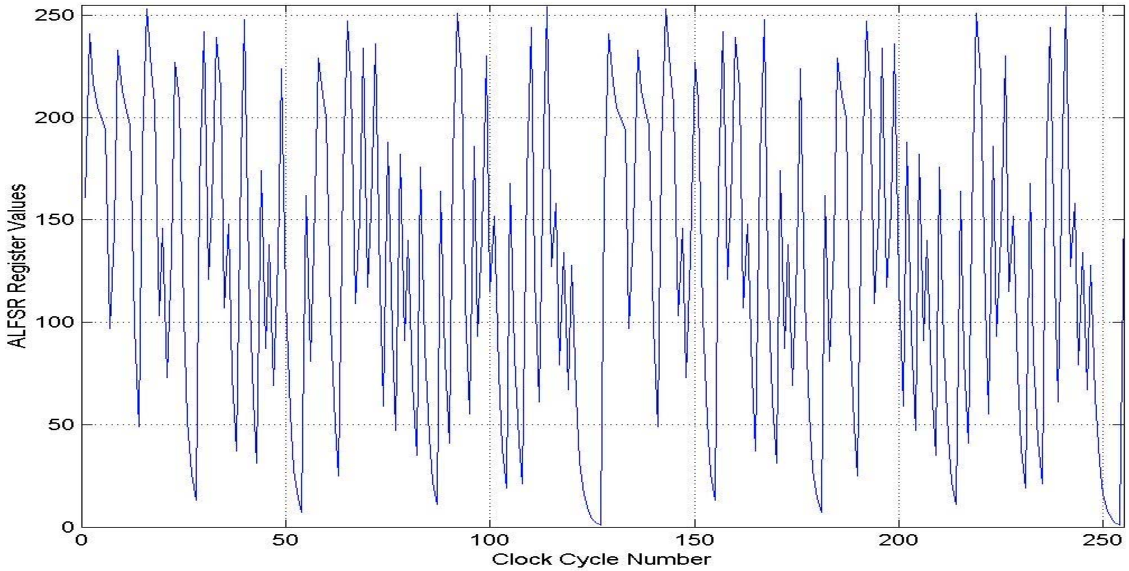
Example - 8 bit random number generator



- $p(x) = 1 + x^2 + x^7 + x^8$
- **Read LFSR tap coefficients from left to right**

Starting from a value of *1* the ALFSR gives the following sequence of values:

1,161, 241,217,205,199,194,97,145,233, 213,203,196,98,49,185,253,223,206,103,…



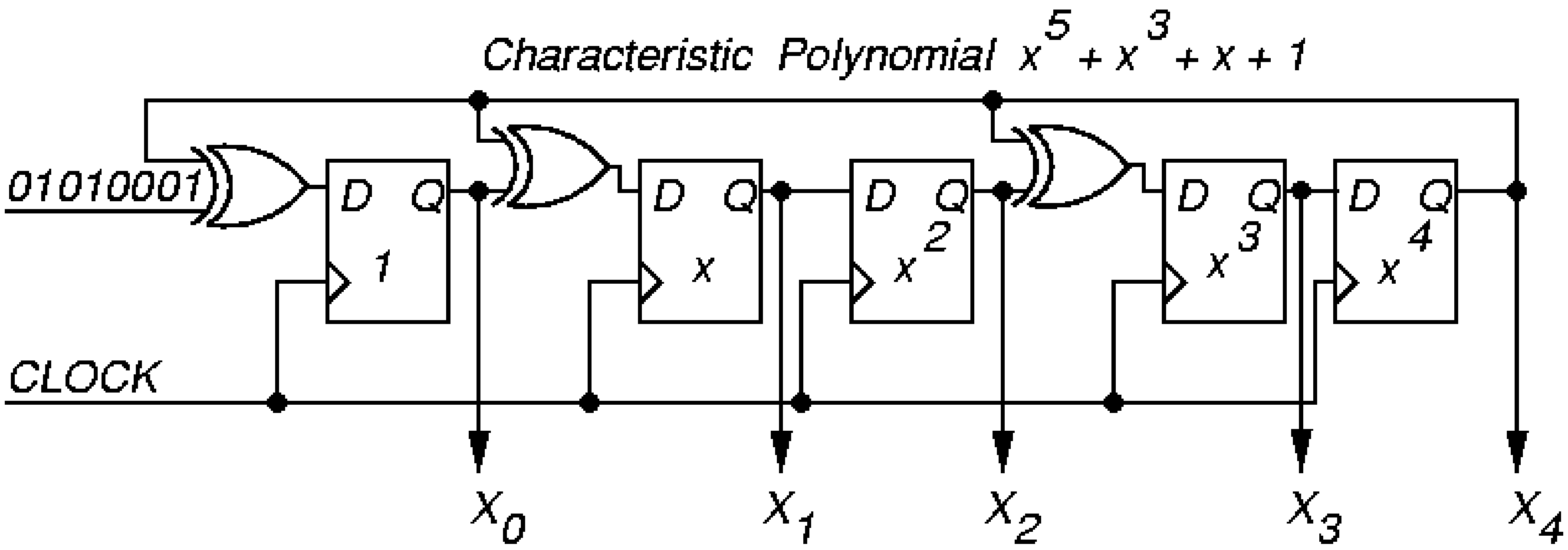
The Need for Signature Analysis

- Severe amounts of data in the response to the LFSR patterns – example:
  - Generate 5 million random patterns
  - Circuit Under Test has 200 outputs
  - Leads to: 5 million x 200 = 1 billion bits response
- Uneconomical to store and check all of these responses on chip
- Responses must be compacted

We can make a signature analyzer by modifying an ALFSR to give it an external input.

The resulting structure is called an *LFSR*.

It can be thought of as a generalization to a parity computation block. In fact, it generates codes (signatures) which are used in so-called ***BCH codes*** used in communication systems.



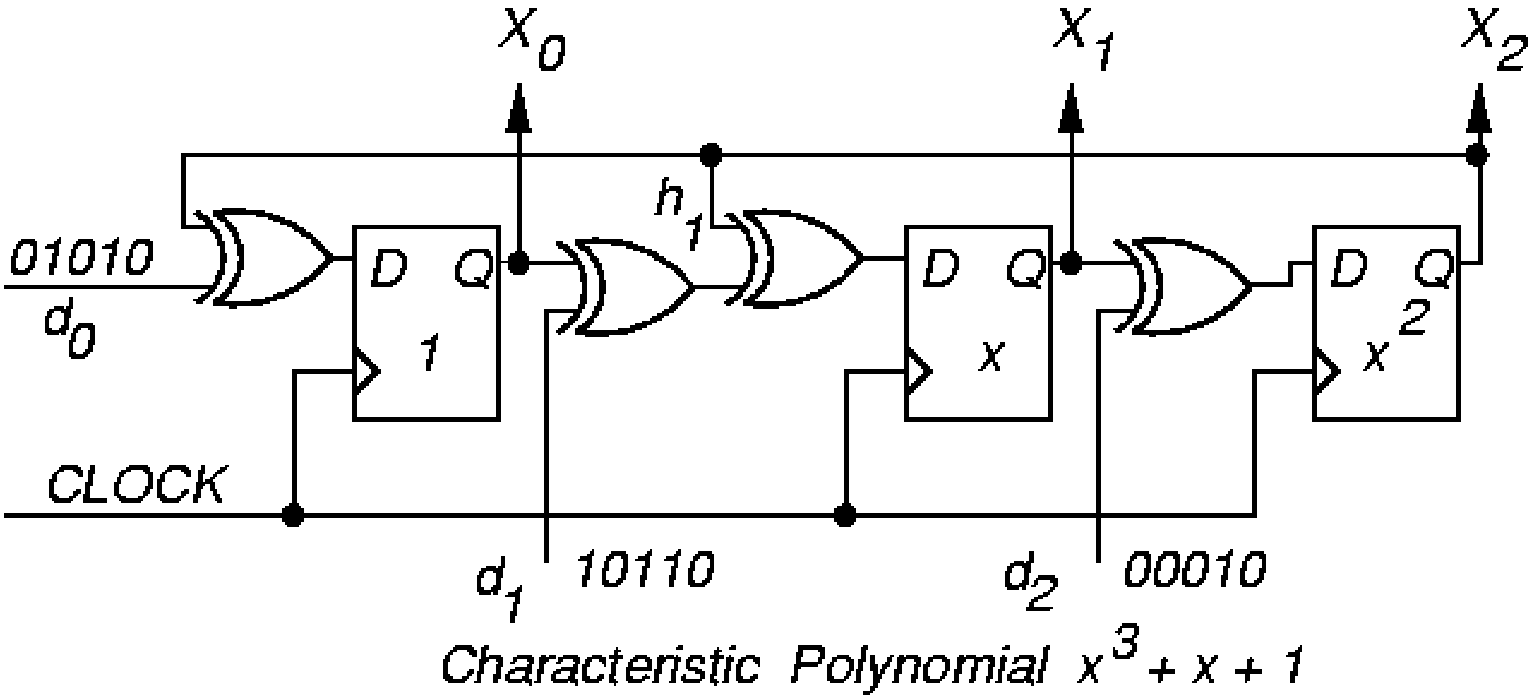
An LFSR. The *signature* is the value of the bits left in the flip-flops after all of the input bits have been shifted in.

Logic Simulation:

Inputs	x^0	x^1	x^2	x^3	x^4
Initial State	0	0	0	0	0
1	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
1	1	0	0	0	1
0	1	0	0	1	0
1	1	1	0	0	1
0	1	0	1	1	0

- **Problem with ordinary LFSR response compacter or signature analyzer:**
  - **Too much hardware if one of these is put on each *primary output* (PO)**

- **Solution: MISR – Multi-Input LFSR**
- **compacts all outputs into one LFSR**
  - **Works because LFSR is linear – obeys *superposition principle***
  - **Superimpose all responses in one LFSR – the final remainder is the XOR sum of the remainders of polynomial divisions of each PO by the characteristic polynomial**



X0
X1
X2
h1
01010
d0
D
Q
1
x
x2
CLOCK
d1
10110
d2
00010
Characteristic Polynomial x3 + x + 1

3-input MISR

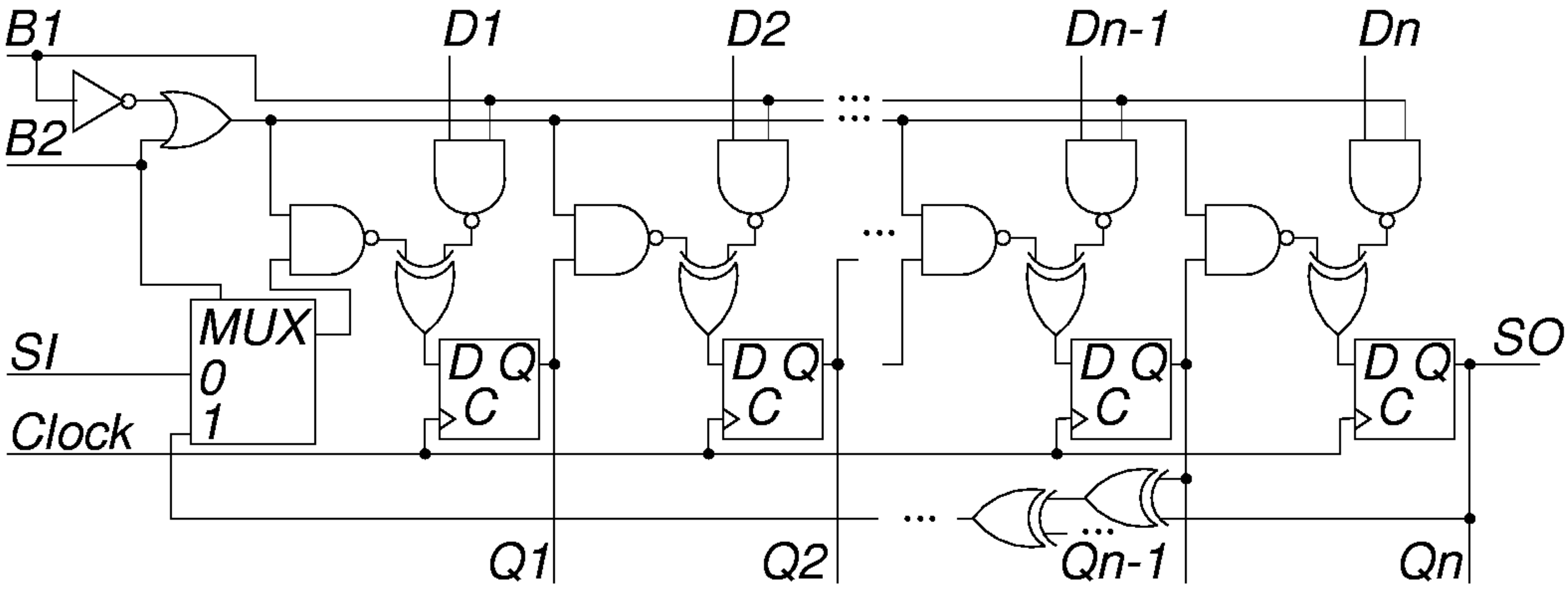
The flip-flops needed for

- normal circuit operation
- scan-path register chain
- ALFSR test vector generation
- LFSR response signature compression

can be all be shared, with some control circuit to specify the usage of the flip-flops.

A circuit that puts it all together is the *BILBO*

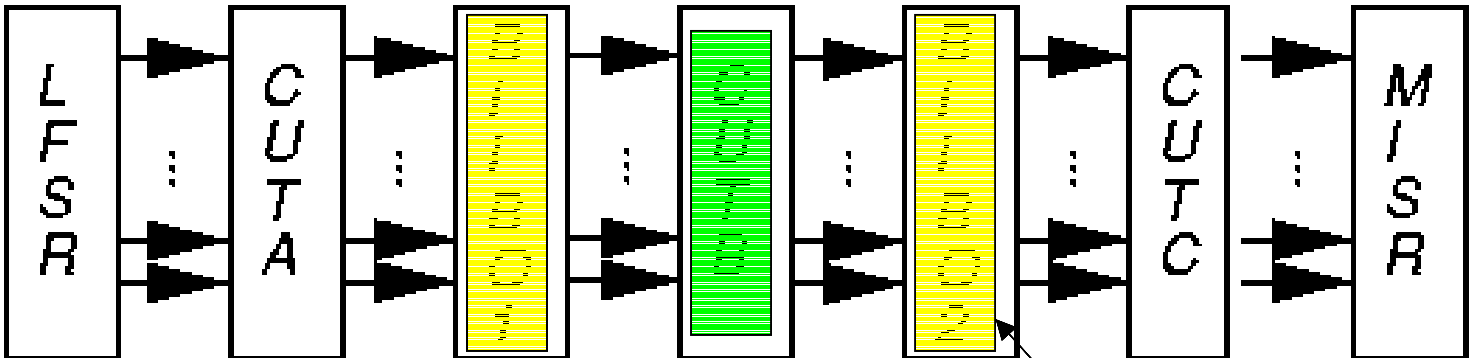
Built-In Logic Block Observer (BILBO)



B. Konemann, *et al., "*Built-In Logic_Block Observation Technique,"
Digest of papers 1979 Test Conf., pp.37-41, Oct., 1979

To Test CUTB:

this BILBO does random test vector generation



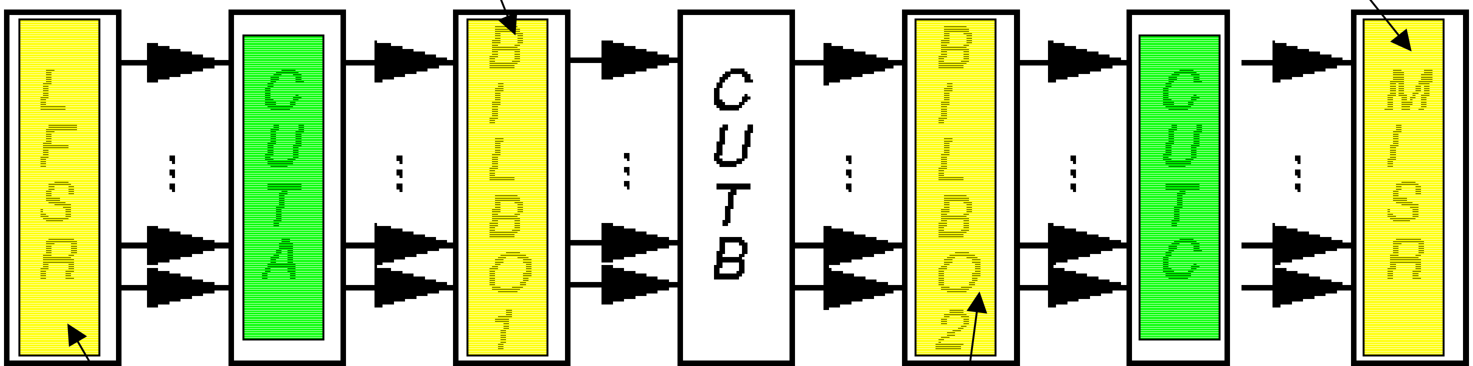
(a) Example test configuration.

while this BILBO does signature analysis

To Test CUTA and CUTC:

this BILBO does signature analysis

this MISR also does signature analysis



(a) Example test configuration.

this LFSR also does random test vector generation

this BILBO does random test vector generation

Boundary Scan approach to *system* test

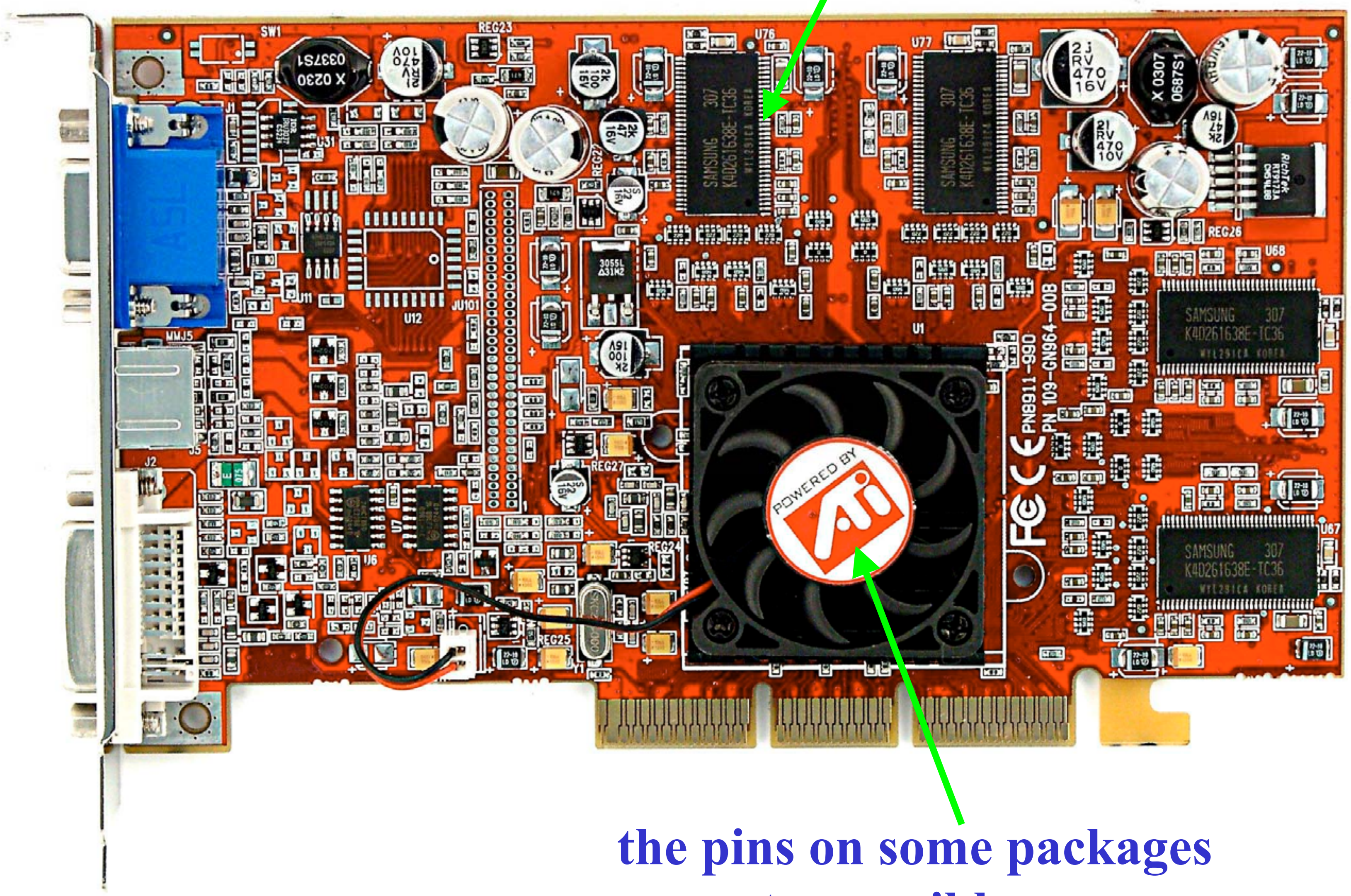
Defined by the ***IEEE 1149.1 JTAG*** Boundary Scan Standard

(JTAG = Joint Test Activities Group)

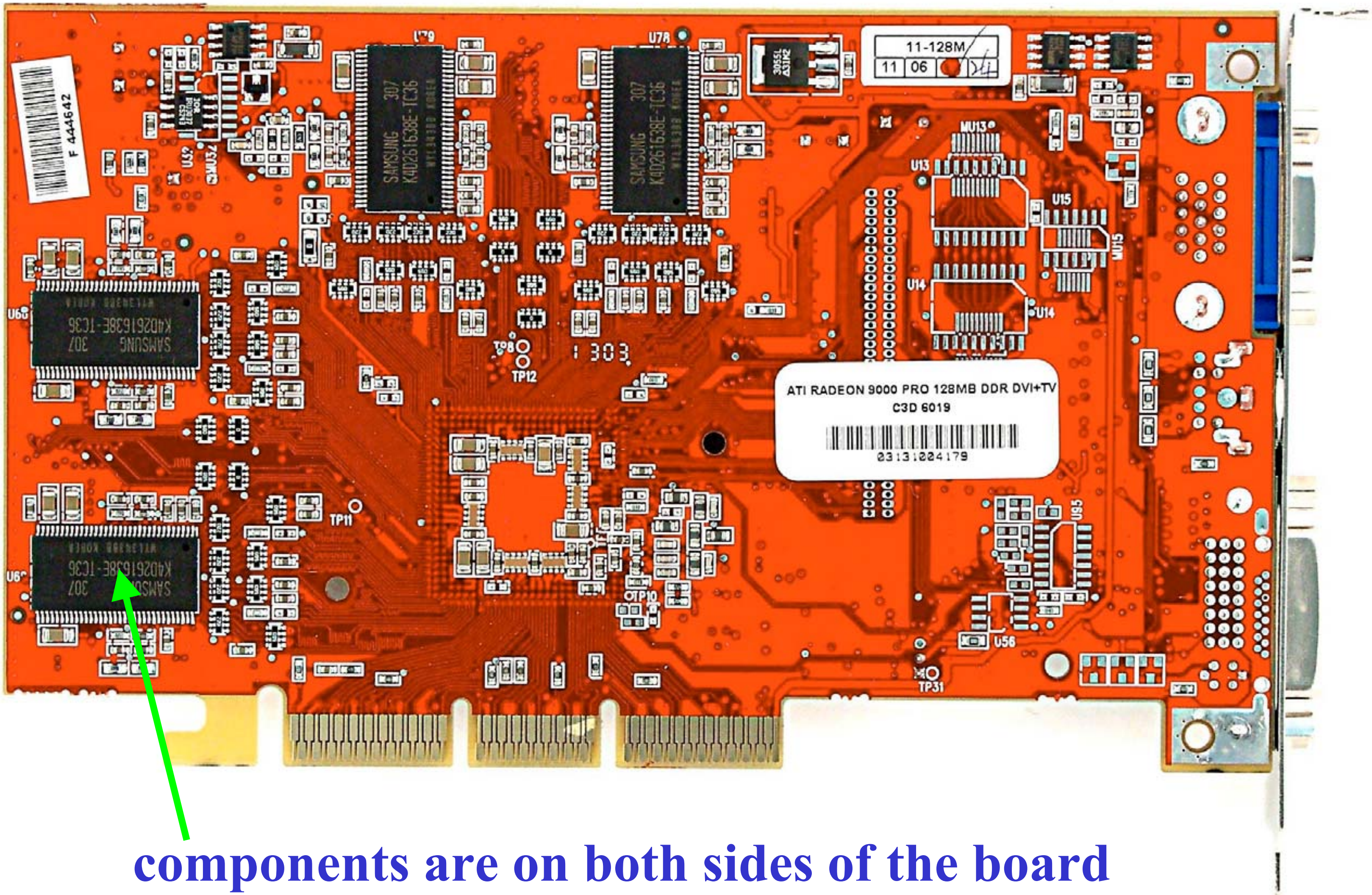
Why is something like Boundary Scan needed?

- ***Bed-of-nails printed circuit board testers are no longer feasible because:***
  - **Components are now on both sides of PCB**
  - **DIP packages have been replaced with flat packs to reduce inductance**
    - **Nails would hit components**
  - **Ball-grid array packages hide the pins under the device**
    - **Nails can't reach the component pins**
  - **Reduced spacing between PCB wires**
    - **Nails would short the wires**

ic pins and pcb traces are very finely packed



the pins on some packages are not accessible



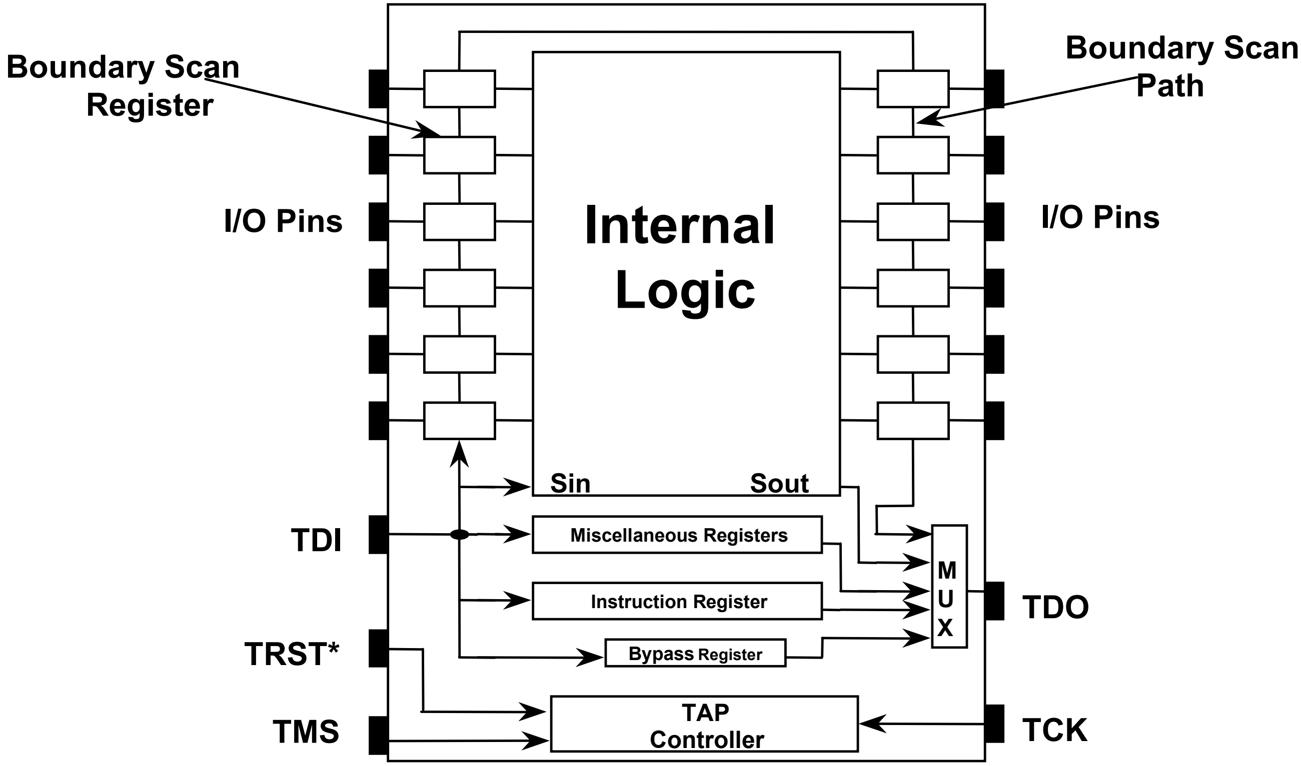
components are on both sides of the board

- ***Therefore we need:***
  - **To replace PCB Testers with built-in test delivery system**
  - **A standard System Test Port and Bus**
  - **To Integrate components from different vendors**
    - **Test bus identical for various components**
    - **One chip has test hardware for other chips**

JTAG 1194 Standard

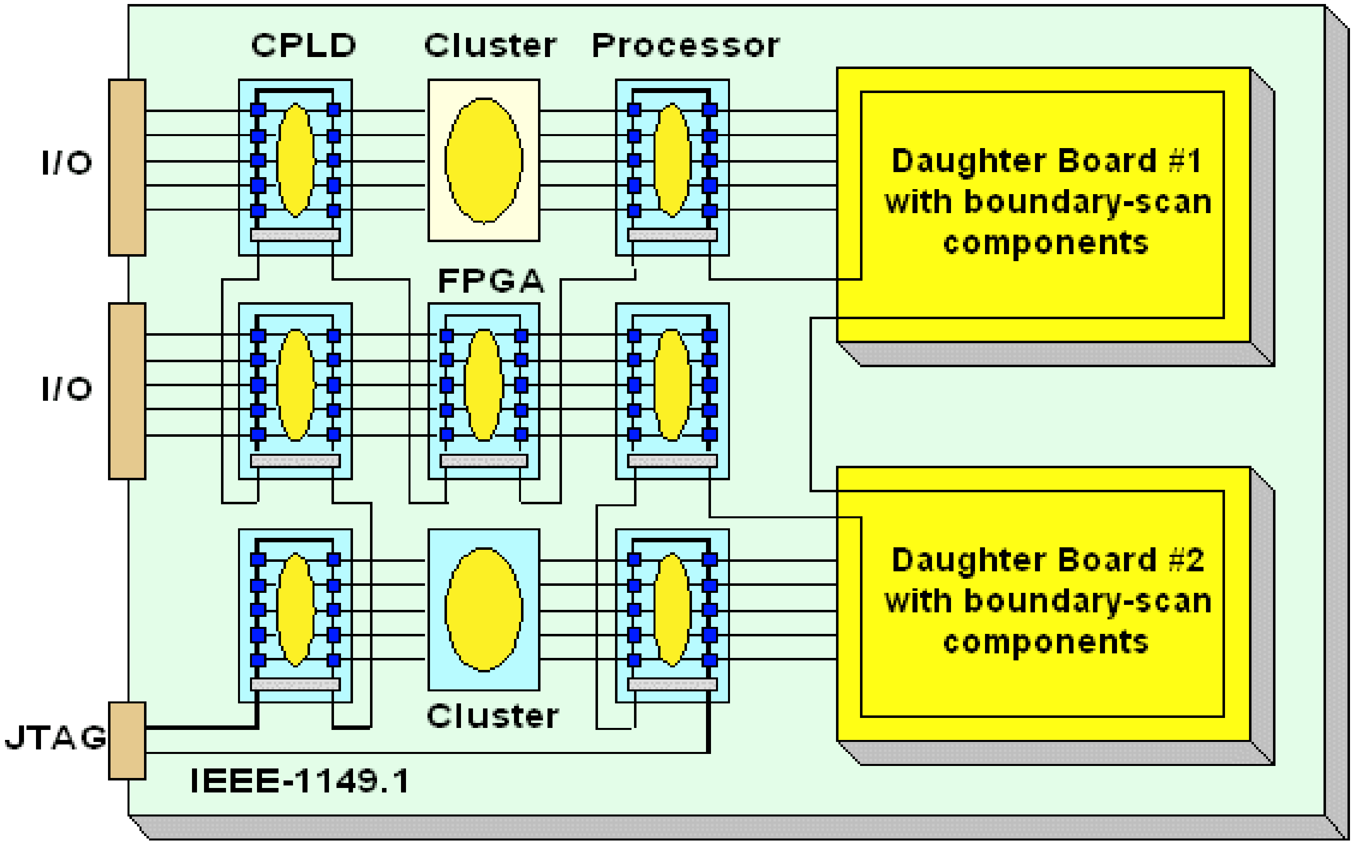
- **Lets test instructions and test data be serially fed into a *component-under-test* (CUT)**
  - **Allows reading out of test results**
  - **Allows *RUNBIST* command as an instruction**
    - **Too many shifts needed to shift in all external tests**
- **JTAG can operate at chip, PCB, & system levels**
- **Allows control of tri-state signals during testing**
- **Lets other chips collect responses from CUT**
- **Lets system interconnects be tested separately from components**
- **Lets components be tested separately from wires**

The JTAG idea is to add a register to every signal I/O pin. These registers can be connected in a single long scan-path.



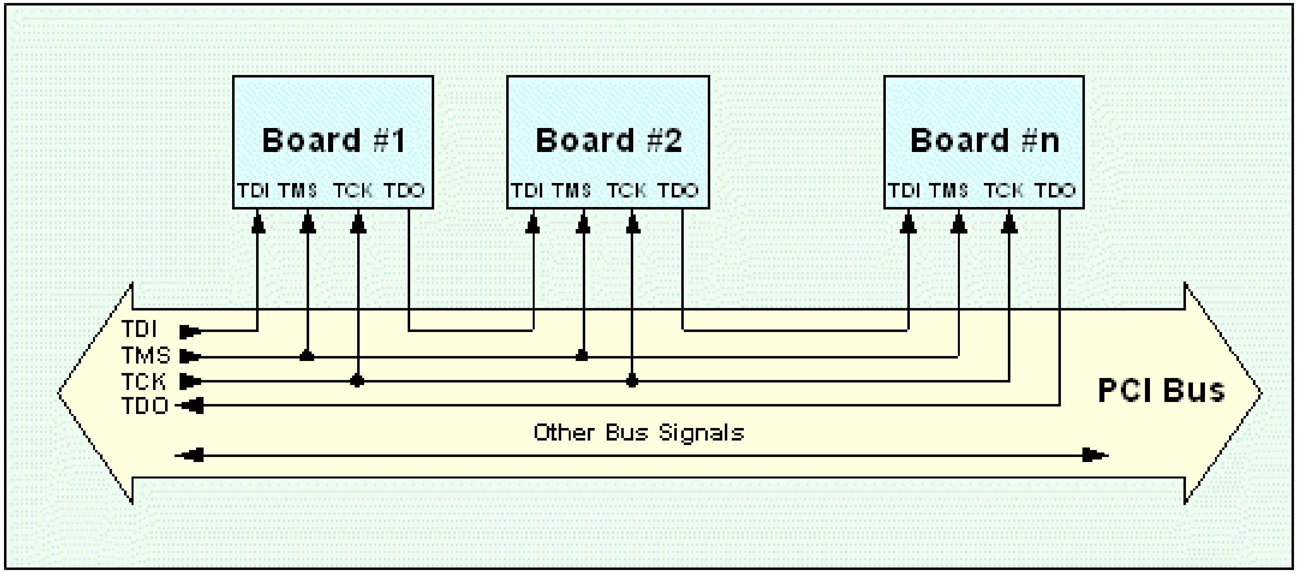
Test data can be read in serially and test responses read out serially through the Test Access Port (TAP) lines TDI and TDO

The boundary scan-paths of many chips can be chained together into one long boundary scan-path.



(diagram is from Corelis Incorporated)

JTAG can be used to test multiple boards connected on a backplane (or motherboard)

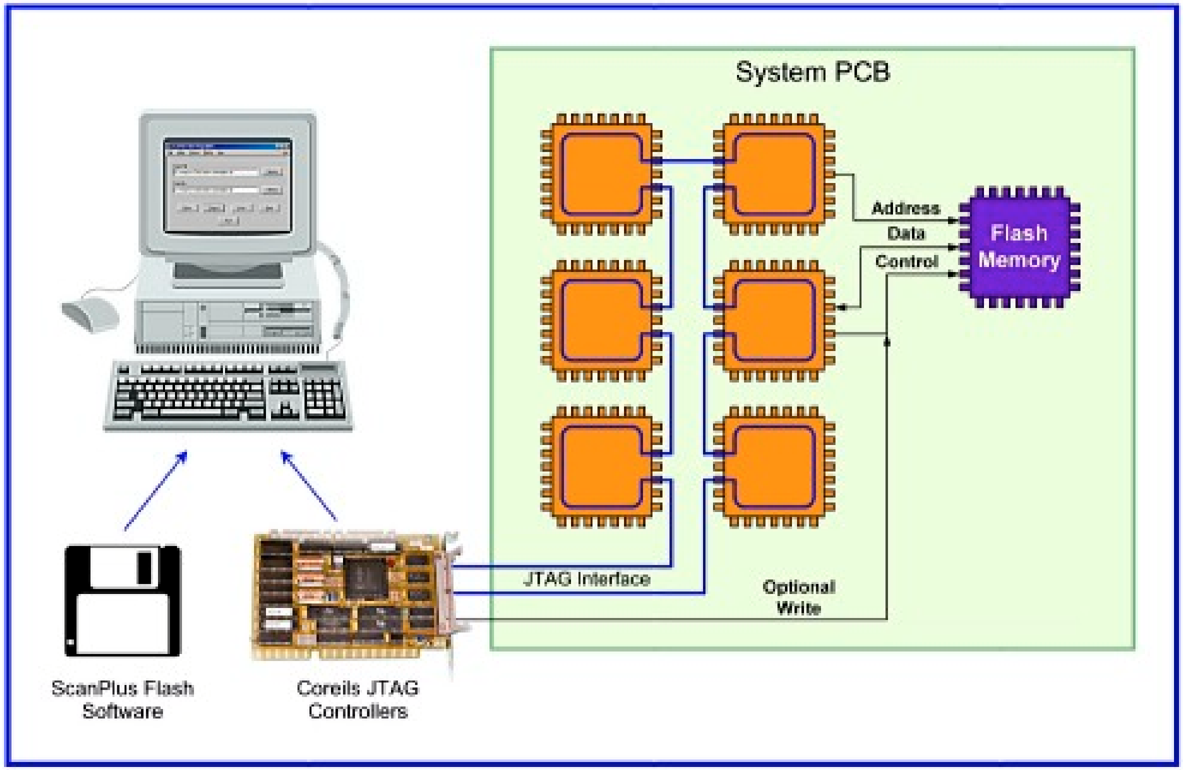


(from Corelis Incorporated)

The JTAG interface is frequently used as a convenient means of entering user data into a chip.

- **Configuration data for FPGAs**
  - the Altera chips on the boards used in the lab can be programmed via JTAG
- **Circuit re-configuration** (over ethernet)
- **Flash memory updating**

In-Circuit programming of Flash memory is becoming a significant application of JTAG interfaces



(from Corelis Incorporated)